Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных средств

**СИСТЕМЫ ОБРАБОТКИ МЕДИАДАННЫХ**

Лабораторная работа № 5

Сжатие изображений

Вариант № 14

группа № 850702

| Выполнил: | Турко В.Д. |
| --- | --- |
| Проверил | Рыбенков Е.В. |

Минск 2021

**1 ЦЕЛЬ РАБОТЫ**

Научиться сжимать изображения на примере формата JPEG.

**2 ЗАДАНИЕ**

1. Выполнить анализ изображения с помощью ДКП, выполнить квантование коэффициентов ДКП, выполнить синтез изображения и оценить качество с помощью psnr.
2. Закодировать коэффициенты ДКП с помощью алгоритма Хаффмана.
3. Выполнить шаги 1 и 2 для b = 1..8 и построить график зависимости psnr от B.

**3 ХОД РАБОТЫ**

Для выполнения лабораторной работы был использован язык программирования Python (v3.9) и изображение, показанное на рисунке 1.

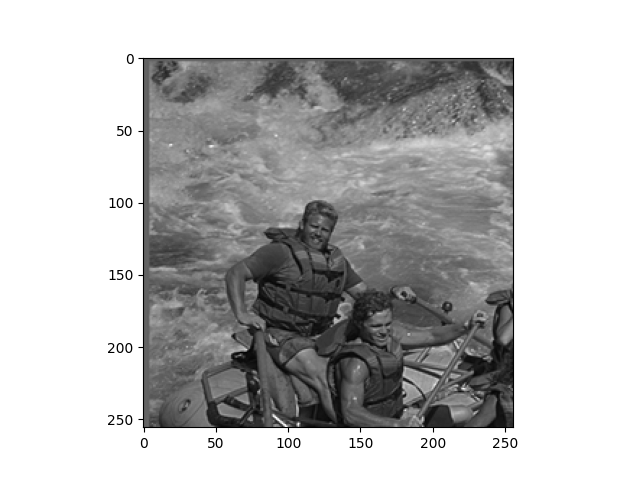


Рисунок 1 - Исходное изображение

**3.1 Функции для квантования ДКП**

Расчет бюджета бит:

epsilon = 1/12

def \_\_round\_bit(init\_b, variance, divider):

result = init\_b + 0.5 \* log2(variance / divider)

return 0 if (result <= 0) else np.round(result, 0)

def bit\_alloc(img, bloc\_shape, init\_b) -> np.ndarray:

bloc\_h, bloc\_w = bloc\_shape

img\_var = np.zeros((img.shape[0] // bloc\_h, img.shape[1] // bloc\_w))

for coord, \_ in np.ndenumerate(img\_var):

y, x = coord

block\_slice = (slice(y \* bloc\_h, y \* bloc\_h + bloc\_h),

slice(x \* bloc\_w, x \* bloc\_w + bloc\_w))

img\_var[y][x] = epsilon \* (np.var(img[block\_slice]) \*\* 2)

divider = np.prod(np.power(img\_var, 1 / img\_var.size))

return np.array([[\_\_round\_bit(init\_b, i, divider) for i in row] for row in img\_var])

Квантование канальных сигналов:

def quantize(img: np.ndarray, bloc\_shape: tuple, init\_b: int) -> np.ndarray:

result = np.zeros(img.shape)

bloc\_h, bloc\_w = bloc\_shape

bit\_budget = bit\_alloc(img, bloc\_shape, init\_b)

for coord, bit in np.ndenumerate(bit\_budget):  
 h = 2 \*\* bit

orig\_y = coord[0] \* bloc\_h

orig\_x = coord[1] \* bloc\_w

block\_slice = (slice(orig\_y, orig\_y + bloc\_h),

slice(orig\_x, orig\_x + bloc\_w))

for block\_coord, value in np.ndenumerate(img[block\_slice]):

b, a = block\_coord

result[orig\_y + b][orig\_x + a] = np.round(value \* h) / h

return result

Расчет psnr:

def psnr(original, compressed):

mse = np.mean((original - compressed) \*\* 2)

if(mse == 0):

return 100

max\_pixel = 255.0

psnr = 20 \* log10(max\_pixel / sqrt(mse))

return psnr

**3.2 Функции для кодирования ДКП**

def zig\_zag(a) -> np.array:  
 diags = [a[::-1,:].diagonal(i) for i in range(-a.shape[0]+1,a.shape[1])]  
 for i, val in enumerate(diags):  
 if (i % 2 == 1):  
 diags[i] = np.flip(diags[i])  
 return np.concatenate(diags)  
  
def run\_length\_encoding(seq):  
 result = []  
 counter = 1  
 current\_symbol = seq[0]  
 for i in range(1, len(seq)):  
 if seq[i] == current\_symbol:  
 counter = counter + 1  
 else:  
 result.append([current\_symbol, counter])  
 current\_symbol = seq[i]  
 counter = 1  
 result.append([current\_symbol, counter])  
 return result  
  
def compress(img, bloc\_shape):

h, w = img.shape

bloc\_h, bloc\_w = bloc\_shape

result = np.zeros(img.shape, dtype='int')

codes = []

for coord, \_ in np.ndenumerate(np.zeros((h // bloc\_h, w // bloc\_w))):

block\_slice = (slice(bloc\_h \* coord[0], bloc\_h \* (coord[0] + 1)),

slice(bloc\_w \* coord[1], bloc\_w \* (coord[1] + 1)))

line = run\_length\_encoding(zig\_zag(img[block\_slice]))

huff\_dict = huffman.codebook(line)

block\_codes = {}

for k, v in huff\_dict.items():

block\_codes[k] = int('0' if v == '' else v, 2)

codes.append(block\_codes)

for block\_coord, val in np.ndenumerate(img[block\_slice]):

result[block\_slice][block\_coord] = block\_codes[val]

return result, codes

**3.3 Код программы**  
  
def get\_image(name):

img = cv.cvtColor(cv.imread(name)[..., ::-1][:256, :256], cv.COLOR\_BGR2GRAY)

return cv.normalize(img.astype('float'), None, 0.0, 1.0, cv.NORM\_MINMAX)

def dct2(x): return dctn(x, norm='ortho')

def idct2(x): return idctn(x, norm='ortho')

init\_bs = np.arange(1,9)

compressed\_bits\_result = np.zeros(init\_bs.shape)

psnr\_result = np.zeros(init\_bs.shape)

img = get\_image('lab5/lena\_gray\_256.tif')

bloc\_shape = (8, 8)

ibloc\_shape = (img.shape[0] // 8, img.shape[1] // 8)

for b in init\_bs:

shuffeled = shuffle(split\_in\_blocks(img, bloc\_shape, dct2), ibloc\_shape)

quantized = quantize(shuffeled, ibloc\_shape, b)

compressed = split\_in\_blocks(shuffle(quantized, bloc\_shape), bloc\_shape, idct2)

psnr\_result[b - 1] = psnr(img, compressed) / 100

encoded, codes = compress(quantized, ibloc\_shape)

compressed\_bits = 0

orig\_bits = 0

for \_, val in np.ndenumerate(encoded):

compressed\_bits += val.item().bit\_length()

for code in codes:

for k, v in code.items():

compressed\_bits += v.bit\_length() + 32

compressed\_bits\_result[b - 1] = compressed\_bits

plt.plot(compressed\_bits\_result, psnr\_result)

plt.show()

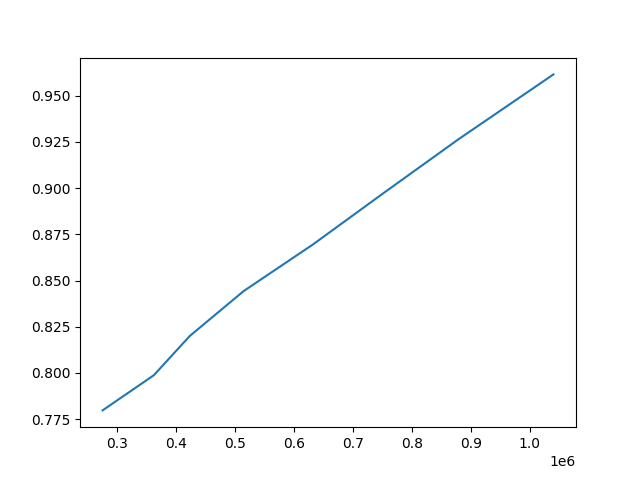


Рисунок 2 - График зависимости PSNR от количества бит, необходимых для кодирования

**4 ВЫВОД:**

В результате выполнения лабораторной работы мной была реализован алгоритм RLE, а также кодирование Хаффмана, позволяющие сжимать изображения.